Assignment 1:-

NAME - SIDDHARTHA ADDY
DEPT. - CSE
YEAR - 3rd (5th Sem)
ROLL NO - 13

1) It is a program that acts as an interface between user (application) and the computer hardware

Functions of OS:-
i) It coordinates the execution of user programs
ii) The primary goal is convinience
iii) The secondary goal is efficiency
iv) It supports the multiple execution modes.

However this depends, because WINDOWS is designed with the primary goal of convinience, but in case of LINUX, the primary goal is efficiency

2) Time sharing or multitasking is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur so frequently that the users may interact with each program while it's running

3) Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor.

4) Spool is a acronym for simultaneous peripherals. Spooling overlaps I/O of one job with the computation of other jobs.

5) A program under execution is known as a process

Process have different states

i) New state – The place where process resides when it is being created or being under creation

ii) Ready state – Many processes reside in the state out of which one will be selected as dispatched

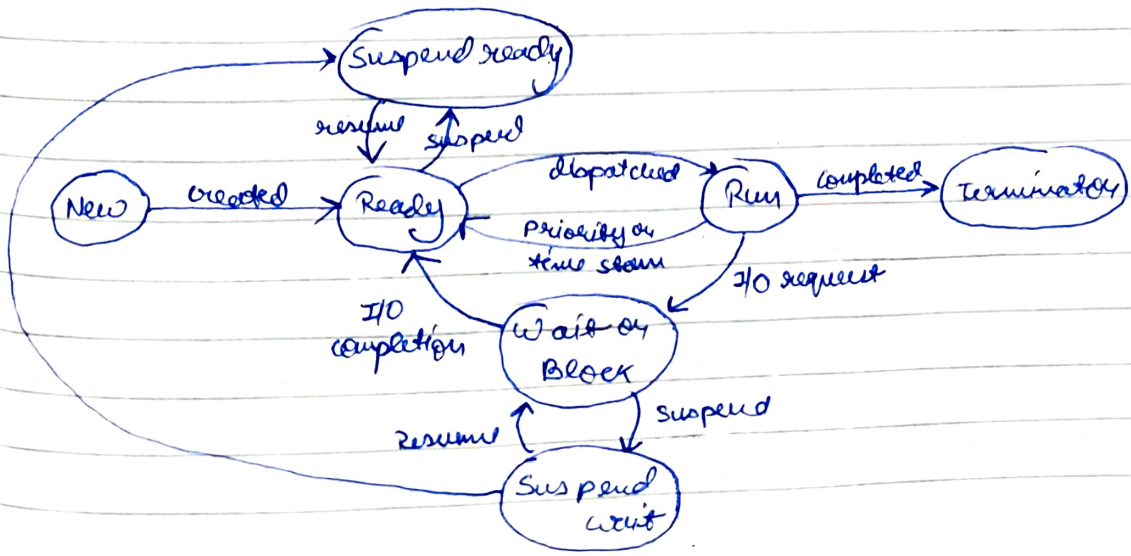
iii) Running State – When the execution of the process takes place

iv) Wait or Block State – Used for I/O operation

v) Termination or Completion state – After the completion of ready, running and wait state the process enters this state

vi) Suspend Ready State – Used to suspend the processes when resources are not available in the ready state

vii) Suspend Block / Suspend Wait state – Used to suspend the processes when resources are not available in wait state.



6) Process Control Block is a data structure that contains information of the process related to it.

It contains various information

i) Process State
ii) Process Number / Process ID
iii) Priority
iv) Program Counter
v) General Purpose Register
vi) List of open files
vii) CPU Scheduling information
viii) ~~CPU~~ ~~MM~~ Memory Management information
ix) Protection informations.

7) In OS there are 3 types of scheduler

i) Long Term Scheduler (LTS) or Job Scheduler
→ LTS is responsible for creating and bringing the new process into the system.

ii) Short Term Scheduler (STS) or CPU Scheduler
→ STS is responsible to select one process in the ready state and scheduling it for CPU

iii) Mid Term Scheduler (MTS)
→ MTS is responsible for suspending and resuming the process.

8) Saving the context of one process and loading the context of another process is known as context switching.

Eg. In Linux kernel, context switching involves switching registers, stack pointers, PC, flushing TLB and loading page Table for next process.

9) Thread is a light weight process

The key differences between Thread & Process

i) Each Thread belongs to one process
ii) Each Thread represents a separate flow of control

Advantages of Thread

i) Responsive - will increase
ii) Faster context switch
iii) Effective utilization of microprocessor system
iv) Resource sharing
v) Enhanced throughput of the system
vi) Economical

10) Based one level there are two types of thread

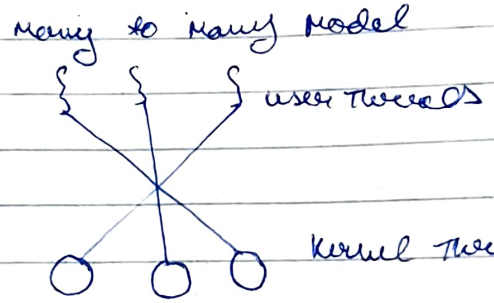
i) User level Thread
ii) Kernel level Thread

User level Thread	Kernel level Thread
→ Implemented by user	→ Implemented by OS
→ Implementation is easy	→ Implementation is hard
→ NO hardware support is requisite	→ Scheduling requires hardware support
→ These are dependent Threads	→ These are independent Threads
→ OS does not recognise user level Threads	→ OS recognises kernel level Threads.

iii) Many OS supports kernel level Thread and user level thread in a combined way. Eg. Solaris.

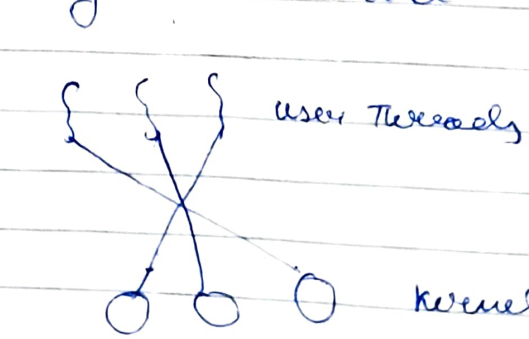
i) Many to Many Model :-

In this model, we have multiple user threads multiplex to same or lesser number of kernel level Threads. Number of kernel level threads are specific to the machine, advantage of this model is if a user thread is blocked we can schedule other user thread to other ~~kernel~~ kernel thread.



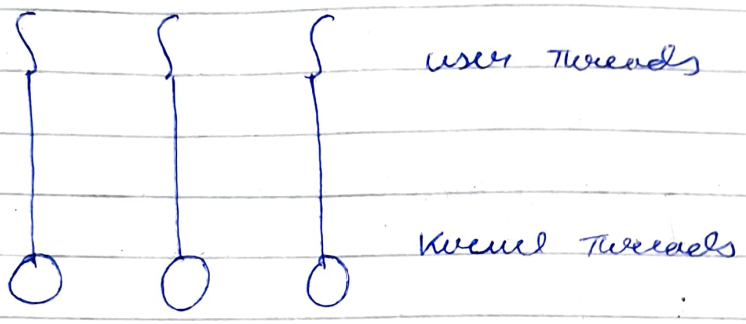
ii) Many to One Model :-

In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks.



11) One to One Model :-

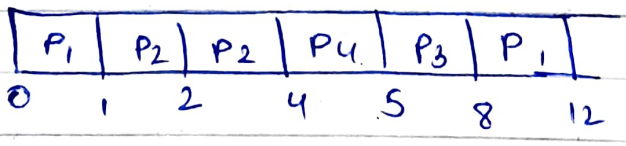
In this model, one to one relationship between kernel and user threads. In this model multiple threads can run on multiple processors. Problem with this model is that creating a user thread requires the corresponding kernel thread.



12)

Process	AT	BT	CT	TAT	WT
P_1	0	8 4 0	12	12	7
P_2	1	3 2 0	4	3	0
P_3	2	3 0	8	6	3
P_4	4	1 0	5	1	0
				22	10

Gantt Chart :-



Avg TAT = 22/4 = 5.5
Avg WT = 10/4 = 2.5
Avg RT = (0+0+3+0)/4 = 0.75

13)

Process	BT	CT	TAT	WT
P_1	~~24 26~~ 0	30	30	6
P_2	~~3~~ 0	7	7	4
P_3	~~3~~ 0	10	10	7
				17

Ready Queue : | ~~P_1~~ | ~~P_2~~ | ~~P_3~~ | P_1 |

Gantt Chart :-

P_1	P_2	P_3	P_1
0	4	7	10	30

Avg WT = 17/5 = 5.67

14)

Process	BT	Priority	CT	TAT	WT
P_1	10	3	16	16	6
P_2	1	1	1	1	0
P_3	2	3	18	18	16
P_4	1	4	19	19	18
P_5	5	2	6	6	1
					41

Gantt Chart :-

P_2	P_5	P_1	P_3	P_4
0	1	6	16	18	19

Avg WT = 41/5 = 8.2

15) Semaphore is an integer variable which is used by various processes in mutual exclusive manner to achieve synchronization

Operations on Semaphore

i) Wait (S) or P : If the semaphore value is greater than 0 decrement the value. Otherwise, wait until the value is greater than 0 and then decrement it

ii) Signal (S) or V : Increment the value of the semaphore

Reader's Writers problem :-

```
sem mutex = 1
sem canwrite = 1
int readcount = 0

write()
{
    while (true)
    {
        down (&canwrite);
        // write data
        up (&canwrite);
    }
}
```

```
reader ()
{
    while (true)
    {
        down (mutex);
        readcount ++;
        if (readcount == 1)
        down (canwrite);
        up (mutex);
        // do the read
        down (mutex);
        readcount --;
        if (readcount == 0)
        up (writer);
        up (mutex);
        // other stuff
    }
}
```

16) Solution 1 : Dining Philosopher's Problem

```
void philosopher ()
{
    while (1)
    {
        Sleep ();
        get_left_fork ();
        get_right_fork ();
        eat ();
        put_left_fork ();
        put_right_fork ();
    }
}
```

However this solution suffers from deadlock

Solution 2 :-

```
#define N 5
#define RIGHT(i) (((i)+1)%N)
#define LEFT(i) (((i)==N) ? 0 : (i)+1)
typedef enum {THINKING, HUNGRY, EATING} phil_state;
phil_state state[N];
semaphore mutex = 1;
semaphore S[N];

void test (int i)
{
    if ( state[i] == HUNGRY &&
         state[LEFT(i)] != EATING &&
         state[RIGHT(i)] != EATING)
    {   state[i] = EATING;
        V(s[i]); }
}
void get_forks (int i)
{       P(mutex);
        state[i] = HUNGRY;
        test(i);
        V(mutex);
        P(s[i]);
}
void put_forks (int i)
{       P(mutex);
        state[i] = THINKING;
        test(LEFT(i));
        test(RIGHT(i));
        V(mutex);
}
```

```
void philosopher(int process)
{
    while(1)
    {   think();
        get_forks(process);
        eat();
        put_forks(process)
    }
```

17) Producer - Consumer Problem :-

```
Semaphore mutex = 1
Semaphore empty = N
Semaphore full = 0

producer()
{
    while (1)
    {
        produce_item(item)
        P(empty);
        P(mutex);
        enter_item(item)
        V(mutex)
        V(full);
    }
}

consume()
{
    while (1)
    {
        P(full);
        P(mutex);
        remove_item(item)
        V(mutex);
        V(empty);
        consume_item(item);
    }
}
```